

Cours 3 - Branching

Stable Maximum

Semestre Automne 2020-2021 - Université Claude Bernard Lyon 1

Christophe Crespelle

`christophe.crespelle@inria.fr`



département

Informatique

Faculté des Sciences et Technologies

Université Claude Bernard Lyon 1

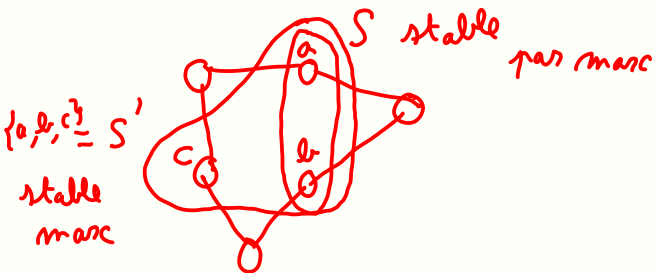
Problème du stable maximum

Cadre : graphes simples, non orientés et sans boucles

Définition

Un stable dans un graphe $G = (V, E)$ est un sous-ensemble $S \subseteq V$ de sommets deux à deux non adjacents (aucune arête entre les sommets de S).

Un stable maximum est un stable S dont le nombre de sommets $|S|$ est maximum (parmi tous les stables de G).



Problème du stable maximum

- **Entrée** : un graphe G
- **Sortie** : un stable maximum de G

Probleme classique : les aretes representent des incompatibilites et on veut trouver un ensemble maximum d'entites compatibles.

Problème du stable maximum

- **Entrée** : un graphe G
- **Sortie** : un stable maximum de G

Probleme classique : les aretes representent des incompatibilites et on veut trouver un ensemble maximum d'entites compatibles.

Exemple : Quel est le nombre maximum de seances de CM/TD/TP qu'un etudiant du M1 peut suivre cette semaine ?

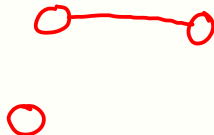
Problème du stable maximum

- **Entrée** : un graphe G
- **Sortie** : un stable maximum de G

Probleme classique : les aretes representent des incompatibilites et on veut trouver un ensemble maximum d'entites compatibles.

Exemple : Quel est le nombre maximum de seances de CM/TD/TP qu'un etudiant du M1 peut suivre cette semaine ?

Les sommets sont les seances de CM/TD/TP de cette semaine et on met une arete entre deux seances si leurs deux plages horaires ont une intersection non vide.



Problème du stable maximum

- **Entrée** : un graphe G
- **Sortie** : un stable maximum de G

Probleme classique : les aretes representent des incompatibilites et on veut trouver un ensemble maximum d'entites compatibles.

Difficulte de calcul : **NP-complet** (\Rightarrow presumption de non existence d'algo polynomial pour le resoudre)

Problème du stable maximum

- **Entrée** : un graphe G
- **Sortie** : un stable maximum de G

Probleme classique : les aretes representent des incompatibilites et on veut trouver un ensemble maximum d'entites compatibles.

Difficulte de calcul : **NP-complet** (\Rightarrow presumption de non existence d'algo polynomial pour le resoudre)

Remarque : pas facile de predire la difficulte d'un probleme comparez Flot Maximum et Stable Maximum, lequel a l'air le plus dur ?

Approche brute force

Algo brute force :

- on considère un par un tous les sous-ensembles de sommets
- pour chacun, on teste si c'est un stable
- on en garde un qui réalise le maximum de la taille

Approche brute force

Algo brute force :

- on considère un par un tous les sous-ensembles de sommets
- pour chacun, on teste si c'est un stable
- on en garde un qui réalise le maximum de la taille



Complexité :

- il y a 2^n sous-ensembles S de V

enumeration $O(n 2^n)$

- tester si S est un stable peut se faire en $O(n + m)$

marquer S

Total : $O((n + m) \cdot 2^n)$, on écrit aussi $O^*(2^n)$

\rightarrow ne tient pas compte des polynômes

$$O(m + m + m) \times O(2^n) = O((m + m) 2^n)$$

Approche brute force

Algo brute force :

- on considère un par un tous les sous-ensembles de sommets
- pour chacun, on teste si c'est un stable
- on en garde un qui réalise le maximum de la taille

Complexité :

- il y a 2^n sous-ensembles S de V
- tester si S est un stable peut se faire en $O(n + m)$

Total : $O((n + m) \cdot 2^n)$, on écrit aussi $O^*(2^n)$

On va faire un algo exponentiel plus rapide ($O(1.45^n)$) en s'appuyant sur 2 propriétés simples, par la technique du branching.

Gain de complexite

On va faire un algo exponentiel plus rapide ($O(1.45^n)$) en s'appuyant sur 2 proprietes simples, par la technique du branching.

$$\frac{2^m}{1.45^m} = \left(\frac{2}{1.45} \right)^m \sim (1.38)^m$$

pour $n=100$

$$1.38^{100} \approx 100\,000\,000\,000\,000$$

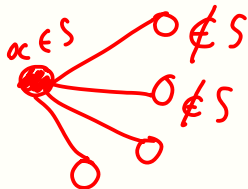
$$\sim 10^{14}$$

En travaillant avec 1.45^m on peut esperer
traiter des graphes où $n=56$ en pratique
ça va être mieux

Premieres proprietes

Propriete

Si S est un stable et $x \in S$ alors $S \cap N(x) = \emptyset$.



Propriete

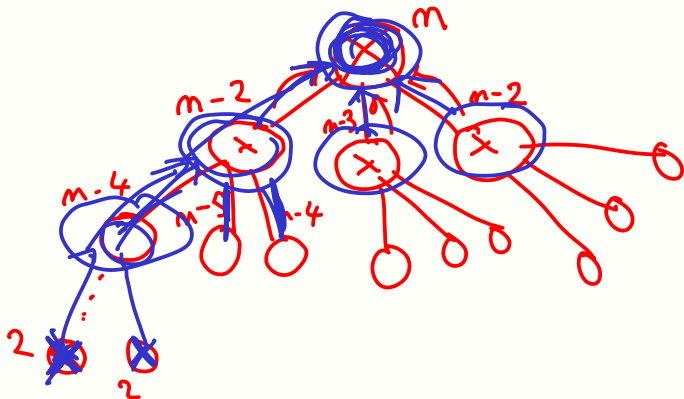
Si S est un stable maximum (donc maximal) et $x \in V$ alors $S \cap (\{x\} \cup N(x)) \neq \emptyset$.



Branching

Idee generale :

- faire une "disjonction" de cas qui tombe sur des problemes de taille reduite
- resoudre **recursivement** chaque probleme reduit
- garder le meilleur resultat



Branching

Pour Stable Maximum, ca donne :

- pour un sommet x on fait un cas pour chaque sommet y dans $\{x\} \cup N(x)$:
 - ▶ on met y dans le stable solution S
 - ▶ on retire y et $N(y)$ de G pour obtenir \tilde{G}_y
 - ▶ on resoud le probleme sur \tilde{G}_y

Branching

Pour Stable Maximum, ca donne :

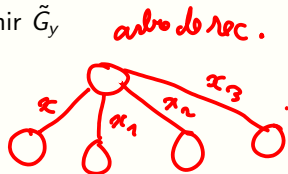
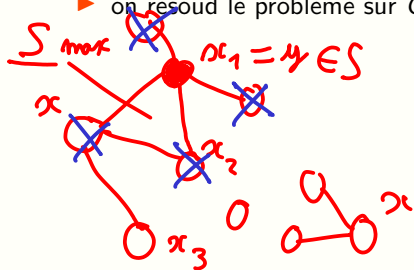
- pour un sommet x on fait un cas pour chaque sommet y dans $\{x\} \cup N(x)$:
 - ▶ on met y dans le stable solution S
 - ▶ on retire y et $N(y)$ de G pour obtenir \tilde{G}_y
 - ▶ on resoud le probleme sur \tilde{G}_y

- pour la solution sur G on prend la meilleure obtenue sur les \tilde{G}_y

Branching

Pour Stable Maximum, ca donne :

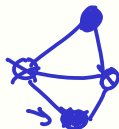
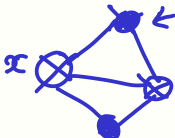
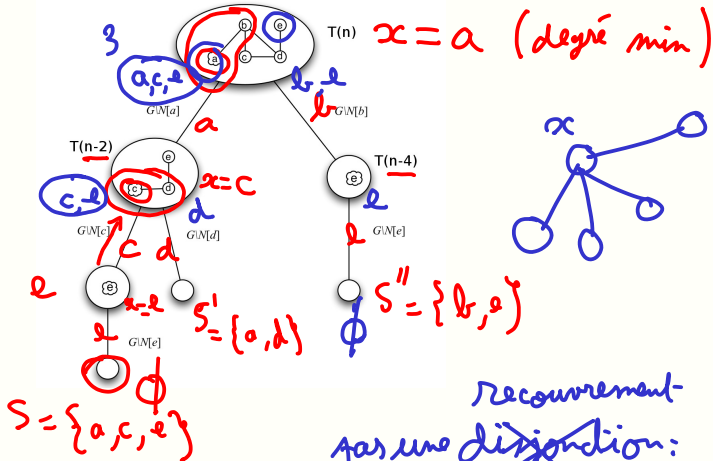
- pour un sommet x on fait un cas pour chaque sommet y dans $\{x\} \cup N(x)$:
 - ▶ on met y dans le stable solution S
 - ▶ on retire y et $N(y)$ de G pour obtenir \tilde{G}_y
 - ▶ on resoud le probleme sur \tilde{G}_y



- pour la solution sur G on prend la meilleure obtenue sur les \tilde{G}_y
- on choisit bien le sommet x avec lequel on fait ca : un sommet de degre minimum

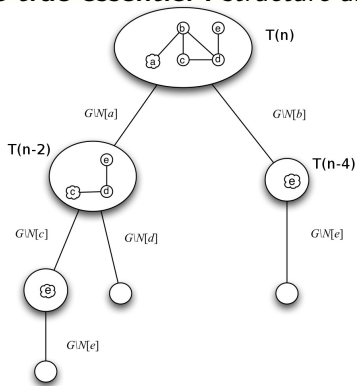
Analyse de l'algorithme

- Le truc essentiel : structure arborescente



Analyse de l'algorithme

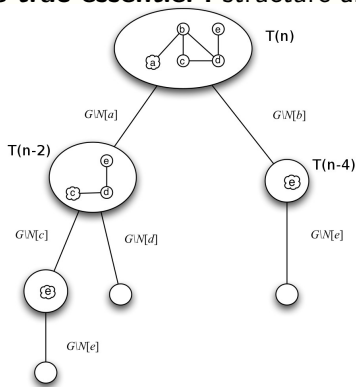
- **Le truc essentiel** : structure arborescente



- **Correction** : OK car on couvre tous les cas

Analyse de l'algorithme

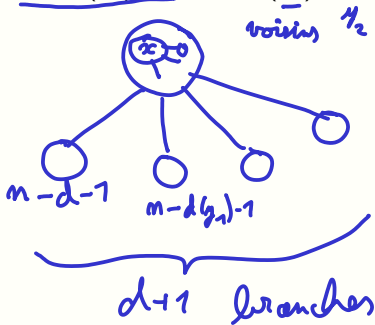
- **Le truc essentiel** : structure arborescente



- **Correction** : OK car on couvre tous les cas
- **Complexite** :
 - ▶ le traitement sur chaque noeud de l'arbre prend $O(n + m)$
 - ▶ complexite totale $O((n + m) \cdot T(n))$
ou $T(n)$ est le nombre de noeuds de l'arbre de recursion,
dans le pire des cas, pour un graphe d'au plus n sommets

Complexite de l'algorithme : majoration de $T(n)$

- lorsqu'on branche sur un sommet x de degre d avec $N(x) = \{y_1, y_2, \dots, y_d\}$ on obtient des graphes a $n - d - 1, \underline{n - d(y_1) - 1}, n - d(y_2) - 1, \dots, n - d(y_d) - 1$ sommets



Complexite de l'algorithme : majoration de $T(n)$

- lorsqu'on branche sur un sommet x de degre d avec $N(x) = \{y_1, y_2, \dots, y_d\}$ on obtient des graphes a $n - d - 1, n - d(y_1) - 1, n - d(y_2) - 1, \dots, n - d(y_d) - 1$ sommets
- de plus comme x est de degre minimum on a $\forall i \in \llbracket 1, d \rrbracket, \underline{d(y_i)} \geq d$

Complexite de l'algorithme : majoration de $T(n)$

- lorsqu'on branche sur un sommet x de degre d avec $N(x) = \{y_1, y_2, \dots, y_d\}$ on obtient des graphes a $n - d - 1, n - d(y_1) - 1, n - d(y_2) - 1, \dots, n - d(y_d) - 1$ sommets
- de plus comme x est de degre minimum on a $\forall i \in \llbracket 1, d \rrbracket, d(y_i) \geq d$

- par monotonicite de $T(\cdot)$, on a donc

$$T(n) \leq 1 + T(n - d - 1) + \sum_{1 \leq i \leq d} T(n - d - 1) =$$
$$1 + (d + 1)T(n - (d + 1))$$

C'est a dire $T(n) \leq 1 + sT(n - s)$, en posant $d + 1 = s$.

Complexite de l'algorithme : majoration de $T(n)$

- lorsqu'on branche sur un sommet x de degre d avec $N(x) = \{y_1, y_2, \dots, y_d\}$ on obtient des graphes a $n - d - 1, n - d(y_1) - 1, n - d(y_2) - 1, \dots, n - d(y_d) - 1$ sommets
- de plus comme x est de degre minimum on a $\forall i \in \llbracket 1, d \rrbracket, d(y_i) \geq d$

- par monotoncité de $T(\cdot)$, on a donc
$$T(n) \leq 1 + T(n - d - 1) + \sum_{1 \leq i \leq d} T(n - d - 1) =$$

$$1 + (d + 1)T(n - (d + 1))$$

C'est a dire $T(n) \leq 1 + (s)T(n - s)$, en posant $d + 1 = s$.

- d'ou, si tous les sommets sur lequel on branche dans l'algo sont de degres d :

$$T(n) \leq 1 + s + s^2 + \dots + s^{n/s} = \frac{1 - s^{1+n/s}}{1 - s} = O(s^{n/s})$$

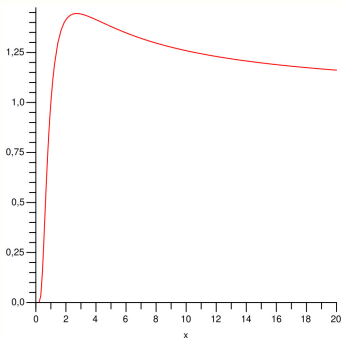
Handwritten notes below the equation:

$$1 + s + (1 + s) + (1 + s)^2 + \dots$$

m/n fois

Complexite de l'algorithme : majoration de $T(n)$

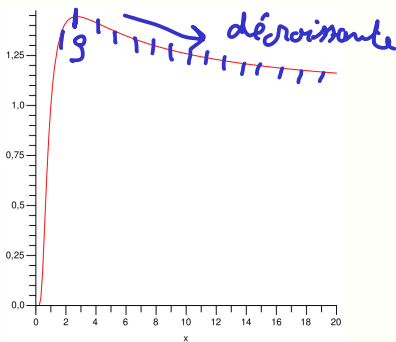
- en etudiant la fonction $s^{1/s}$, on s'aperçoit qu'elle admet un maximum sur les s entiers pour $s = 3$, on obtient alors $T(n) = O(3^{n/3})$ soit $T(n) = O(1.45^n)$...



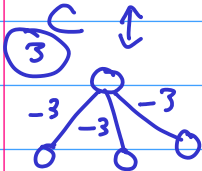
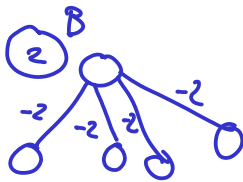
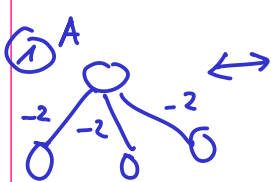
- ... si on ne branche que sur des sommets de meme degre !!!!

Complexite de l'algorithme : majoration de $T(n)$

- en etudiant la fonction $s^{1/s}$, on s'aperçoit qu'elle admet un maximum sur les s entiers pour $s = 3$, on obtient alors $T(n) = O(3^{n/3})$ soit $T(n) = O(1.45^n)$...



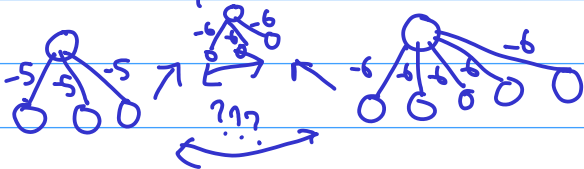
- ... si on ne branche que sur des sommets de meme degre !!!!
- ca reste vrai dans le cas general car on a choisi le "pire" $d = \cancel{2}$
 $n = d + 1$ $n = 3$



c'est quoi le mieux ?

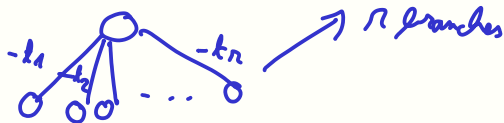
C mieux que A

A mieux que B



Un cadre general pour l'analyse des algo de branching

- **Branching vector.** Si le branchement sur un graphe a n sommets produit un graphe a $n - t_1$ sommets, un a $n - t_2$, ..., un a $n - t_r$, le branching vector de la regle est (t_1, t_2, \dots, t_r)



Un cadre general pour l'analyse des algos de branching

- **Branching vector.** Si le branchement sur un graphe a n sommets produit un graphe a $n - t_1$ sommets, un a $n - t_2$, ..., un a $n - t_r$, le branching vector de la regle est (t_1, t_2, \dots, t_r)
- **Branching factor.** Si tous les branchements sont les memes on se retrouve a resoudre :

$$T(n) \leq T(n - t_1) + T(n - t_2) + \dots + T(n - t_r)$$

Un cadre general pour l'analyse des algos de branching

- **Branching vector.** Si le branchement sur un graphe a n sommets produit un graphe a $n - t_1$ sommets, un a $n - t_2, \dots$, un a $n - t_r$, le branching vector de la regle est (t_1, t_2, \dots, t_r)
- **Branching factor.** Si tous les branchements sont les memes on se retrouve a resoudre :

$$T(n) \leq T(n - t_1) + T(n - t_2) + \dots + T(n - t_r)$$

Maths $\Rightarrow T(n) \leq \alpha^n$ ou α est l'unique racine reelle du polynome $x^n - x^{n-t_1} - x^{n-t_2} - \dots - x^{n-t_r}$.

α est appele le branching factor de (t_1, t_2, \dots, t_r) , note $\tau(t_1, t_2, \dots, t_r)$

α^n

Un cadre general pour l'analyse des algos de branching

- **Branching vector.** Si le branchement sur un graphe a n sommets produit un graphe a $n - t_1$ sommets, un a $n - t_2$, ..., un a $n - t_r$, le branching vector de la regle est (t_1, t_2, \dots, t_r)
- **Branching factor.** Si tous les branchements sont les memes on se retrouve a resoudre :

$$T(n) \leq T(n - t_1) + T(n - t_2) + \dots + T(n - t_r)$$

Maths $\Rightarrow T(n) \leq \alpha^n$ ou α est l'unique racine reelle du polynome $x^n - x^{n-t_1} - x^{n-t_2} - \dots - x^{n-t_r}$.

α est appele le branching factor de (t_1, t_2, \dots, t_r) , note $\tau(t_1, t_2, \dots, t_r)$

- **Maths** \Rightarrow si dans T on a des branchements avec des branching vector differents, on a quand meme $T(n) \leq \alpha^n$ avec α le plus grand (donc le pire) des branching factor dans T

Tableau 1

Ra: par cet algo on a en fait tous
les stables maximum

⚠ Pb: on trouve plusieurs fois chaque stable
max



on le fait au TP2

Tableau 2

Tableau 3